Variables

This tutorial describes iovar, the basic variable object for ioapiTools.   Specifically, you will learn how to extract a variable from a file, how to read general metadata info from the variable, and how to access iovar's axes.

---

1. Extracting a variable
The following assumes that you have already opened a file (see "Opening a File"):

```
## list the variables available in the file
f = listvariables()
##  ['O3', 'CO', 'NO2', 'NO']


## extract no2 and co from the IOAPI file and
## o3 from the CF netCDF file
no2 = f("no2")
co = f("co")
o3 = g("o3")
```

If you want to extract a variable that spans multiple files, use the iofilescan object (see "Opening a File"):

```
## o3 spans 3 files
o3_long = fs("o3")
```

The 'o3' variable, as well as the other variables, is an iovar object, the major repository for data in the ioapiTools package. In addition to storing the o3 data for this domain in a Numeric array, the iovar variable has a whole series of attributes that store metadata, including projection info, units, and spatial and temporal domains, to name a few. The iovar objects also have a long list of methods. Many of these methods are inherited from the cdms temporary variable, the parent class of the iovar object, but there are also many iovar specific methods that will be explored in this tutorial and in subsequent tutorials.

. Contents

---

2. General Metadata

Accessing metadata is essential for effectively analyzing data:

```
## Get general info on size of variable
o3
##
 name : O3
 data: array (24, 1, 70, 58), type = f, 97440 elements
 projection: Lambert Conformal Conic
 object: iovar
##
o3_long
##
 name : O3
 data: array (59, 1, 70, 58), type = f, 239540 elements
 projection: Lambert Conformal Conic
 object: iovar
##
```

By just typing an iovar variable's name, you will get some key pieces of information: name, array dimensions, projection type, and object type. The array dimensions correspond to time, level, latitude (or row), and longitude (or column). In addition, you can see that the array data is a Numeric type "f" or float, and the total number of elements in the array. Here, the only difference between 'o3' and 'o3_long' is the time dimension, 'o3' spans 24 hours and 'o3_long' spans 59 hours.

To get a much more detailed description of a variable's metadata, you should use the *info* method:

```
## Dump all the metadata
co.info()
##
 id: CO
 shape: (24, 1, 70, 58)
 filename:
 missing_value: [  1.00000002e+20,]
 comments:
 grid_name: <None>
 grid_type: generic
 .
 .
 .
##
```

. Contents

3. Axes

One of the key strengths of the iovar object is that every variable has a series of physical axes associated with the data. These axes describe the data's "location" in the temporal and spatial domains. The axes are objects themselves and have their own attributes and methods. The axis object has been directly inherited from the cdms object without modification (see CDAT's information on axes for more info).

Time Axis

```
## get the time axis,
## because we are not capturing it, it will simply
## print some info to the screen
o3.getTime()
##
   id: time
   Designated a time axis.
   units:  hours since 1996-06-24 06:00:00.00
   Length: 24
   First:  0.0
   Last:   23.0
   Other axis attributes:
      calendar: gregorian
      axis: T
   Python id:  42691f0c
##

## save time axis
## return a list of times, in cdtime object form
timeAxis = o3.getTime()
```

```
timeAxis.asComponentTime()
##
 [1996-6-24 6:0:0.0, 1996-6-24 7:0:0.0, 1996-6-24 8:0:0.0,
 1996-6-24 9:0:0.0, 1996-6-24 10:0:0.0, 1996-6-24 11:0:0.0,
 1996-6-24 12:0:0.0, 1996-6-24 13:0:0.0, 1996-6-24 14:0:0.0,
 1996-6-24 15:0:0.0, 1996-6-24 16:0:0.0, 1996-6-24 17:0:0.0,
 1996-6-24 18:0:0.0, 1996-6-24 19:0:0.0, 1996-6-24 20:0:0.0,
 1996-6-24 21:0:0.0, 1996-6-24 22:0:0.0, 1996-6-24 23:0:0.0,
 1996-6-25 0:0:0.0, 1996-6-25 1:0:0.0, 1996-6-25 2:0:0.0,
 1996-6-25 3:0:0.0, 1996-6-25 4:0:0.0, 1996-6-25 5:0:0.0]
##
```

The *getTime* method returns the time axis. In the second case, we are assigning that to a variable, "timeAxis".   Then we are using the axis method *asComponentTime* to return the data in the time axis in cdtime format. In other words, we now have a list of dates, which corresponds to each temporal slice of the data.

Spatial Axes

```
## get the level or layer axis (vertical axis)
## Not so interesting, seeing that aconc has only 1 layer
o3.getLevel()
##
   id: layer
   Designated a level axis.
   units:
   Length: 1
   First:  1.0
   Last:   1.0
   Other axis attributes:
      positive: down
      var_desc: sigma levels
      standard_name: atmospheric_sigma_coordinate
      axis: Z
   Python id:  4272412c
##

## get the latitude axis
o3.getLatitude()
##
   id: yLat
   Designated a latitude axis.
   units:  meters
   Length: 70
   First:  -378000.0
   Last:   450000.0
   Other axis attributes:
      long_name: y coordinate of projection
      standard_name: projection_y_coordinate
      axis: Y
      topology: linear
   Python id:  4272416c
##

## get longitude axis and get a list of all its values
lonAxis = o3.getLongitude()
lonAxis.getValue()
##
[-270000.,-258000.,-246000.,-234000.,-222000.,-210000.,-198000.,-186000.,
      -174000.,-162000.,-150000.,-138000.,-126000.,-114000.,-102000., -90000.,
       -78000., -66000., -54000., -42000., -30000., -18000.,  -6000.,   6000.,
        18000.,  30000.,  42000.,  54000.,  66000.,  78000.,  90000., 102000.,
       114000., 126000., 138000., 150000., 162000., 174000., 186000., 198000.,
       210000., 222000., 234000., 246000., 258000., 270000., 282000., 294000.,
       306000., 318000., 330000., 342000., 354000., 366000., 378000., 390000.,
       402000., 414000.,]
##
```

The horizontal axes are in the native coordinate system. I use the term "yLat" to refer to the Latitude and "xLon" to refer to the Longitude in these native coordinates. In this case, xLon and yLat are in a Lambert Conformal Conic projection, which means that the native coordinates are in meters from a projection center. For example, the xLon value of −270,000 means that this cell is 270 km West of the projection center. These values correspond to the location of the center of the grid cell. In this native coordinate system, the grid cells are a regular grid of equally spaced cells, in this case 12km x 12km (see "Projection and IOAPI Metadata" tutorial for more details.

To get the location of the boundaries of the grid cells:

```
## Get the boundaries of the first 2 grid cells
lonAxis.getBounds()[0:2]
##
 [[-276000.,-264000.,]
  [-264000.,-252000.,] ]
##
```

Contents Previous Next